IPARP Table

IP global data structure

Wed, June 29, 1992

Every network node that supports Internet Protocol (IP) communications maintains an ARP table to relate IP addresses to hardware addresses. In the case of the local station's IP support, it also provides the "node#" that is used to reference that node (if it is a front end data source), and the port#s in use by that node. A "pseudo node#" is used to refer to a port of a node. It provides a functionality similar to that of an internet socket. This note describes how the IPARP table is used by the local station system to support IP.

```
Procedure InzIPARP;
```

At reset time, if no 'IP' exists in the first word of the IPARP table, which is system table#28, initialize the table and clear all entries; otherwise, clear the port# block ptrs in all entries. Although IP addresses are saved across resets, port# assignments for pseudo node#s are not. If the table header includes a nonzero IP address and netMask, then establish a ptr to the IPARP table in a system global variable. Note that when the table is first established and all entries are cleared, the header is also cleared. The local station's IP address and network mask must be entered manually and the system reset again, in order to enable IP support. See installation section.

```
Function PsNIPARP(port: Integer; ipA: Longint; VAR netA: NetAddrType):
Integer;
```

This function is used when an IP (or ARP) datagram is received and processed by the SNAP task. If it is a UDP datagram, the source port# is specified in the call; otherwise, port#0 is used. The table is searched for a match on the source IP address (or sender's IP address in the case of an ARP message), and the hardware address is updated there. If it is not in the table, a new entry is added to the table to hold this information. If there is no allocated port# block for the entry, then one is allocated. For UDP, the port# is installed in the port# block, if it is not already there. The returned value is the pseudo node# that is used to reference the entry. The format of the pseudo node# is chosen so that it does not conflict with node#s in current use among accelerator nodes. This uniqueness is used to denote that IP encapsulation is needed when sending the message on the network, and by reference to the IPARP table entry, the parameters of that encapsulation.The current form of the pseudo range is 1–15.) In the case of an error, a zero is returned. The IPARP table entry is 16 bytes as follows:

NETADDR	NODE#
IPADDR	^IPPORTS

```
Function GetIPARP(pNode: Integer; VAR port: Integer; VAR node: Integer;
VAR ipA: Longint; VAR netA: NetAddrType): Integer;
```

When a frame is due to be transmitted, and the frame header is being built, this function retrieves the IP address, network address, node# and port#, from the IPARP table entry specified by the pseudo node#. These values are used to build the IP header and UDP header of the datagram. The node# is used to replace the pseudo node# used as the destination node field in the acnet header and classic protocol cases.

```
Function IncIPARP(pNode: Integer): Integer;
Function DecIPARP(pNode: Integer): Integer;
```

For each port# registered in the port# block associated with a given IP node, there is a use count. These two functions operate on the use count pointed to by the given pseudo node#, which provides for support of multiple requests from a given UDP source port. When a data request is initialized, and a ptr to its request memory block is inserted into the chain of active requests, IncIPARP is used to advance the use count associated with the target port for the reply. When the request is cancelled, and the request is removed from the active chain, then DecIPARP is used to reduce the use count. In this way, the available 15 port# slots associated with an IP node can be reused as needed.

```
Function NodIPARP(pNode: Integer; node: Integer): Integer;
```

The concept of a node# word is used in both acnet and classic data request protocols. At the time PsNIPARP is called by the SNAP task, any node# specified in the message is unknown. When higher level acnet protocol handling determines what the source node# is, then it can use this function to update the node# word in the IPARP table entry.

```
Procedure TimIPARP;
```

Every second, the QMonitor Task calls this routine to perform timeout logic on IP communications. TimIPARP scans all active IPARP entries and counts down the timeout word in the port# block header. The timeout word is reset to a large value (currently 4000 seconds) every time the port# block is referenced by PsNIPARP or GetIPARP calls. Thus, when it reaches zero, it means no ports have been used for a long time with that IP node, so the block is released.

With the implementation of fragmentation and reassembly, additional timeout logic has been included in TimIPARP. When IP fragments are received, they are timed out in case not all fragments are received to make a complete datagram. Each time a fragment is received that is part of a given datagram, a timeout count is reset to a large value, which is currently 60 seconds. After that time, the fragment blocks are released, and a "time exceeded" ICMP error message is returned to the sending host.

When all IP fragments have been received for a given datagram, a complete datagram is built and passed to the SNAP Task via its message queue, just as if the entire datagram were received from the network. The block containing the complete datagram is timed out in case its associated message counter is not reduced to zero. (The message count word should be decremented by every program or task when it is finished processing the message; this can be done automatically by NetRead, NetRecv, UDPRead, UDPRecv, or by Classic Task processing, depending on the protocol used.) The current value for timing out completed datagram blocks that are unused is 60 seconds. Normally, the datagram block is released within one second after the message count word is reduced to zero by the next call to TimIPARP.

```
Function FrgIPARP(pNode: Integer; dIdent: Integer;
   fragPtr: FBlkPtr; VAR dgPtr: DgPtrType); Integer;
```

This routine is called by the SNAP Task to handle IP fragment processing. The `pNode` parameter is the pseudo node# returned from a call to PsNIPARP. The `dIdent` is the identification field from the fragment IP header that identifies the datagram of which it is only a fragment. The `fragPtr` argument is a ptr to a fragment block that holds a copy of the received fragment. The `dgPtr` variable is set to a ptr to the completed datagram block in the case that this fragment makes the datagram complete. See the document Fragmentation and Reassembly for more details on this.

The IPARP table header format is as follows:

IPADDRS	NETMASK
DEFGATE	MTULOC | MTUEXT

The local station IP address must be entered into the table manually. Since IPARP is in non-volatile memory, it should not thereafter need to be changed. In addition, the subnet mask is kept there and an IP address of the optional default gateway. When a datagram is to be sent to an IP address destination for which there is no known hardware address,, a check is made using the local station IP address and the subnet mask to determine whether the target IP node is on the same subnetwork. If it is, then an ARP request message is sent to obtain the hardware address. If it is not, then the default gateway IP address is used to look up the gateway's hardware address. (If there is none as yet, an ARP request is sent to obtain it.) The message is then sent to the gateway's hardware address.

At the time this is written, when an ARP request must be sent, the datagram to be sent to the target node is discarded. Since the IPARP table entries are not timed out, this is not expected to be a problem. If it is, then a means of queuing up, and timing out, datagrams awaiting ARP responses must be found. With no means of timing out IPARP entries, it is possible for stale information to accumulate there. If a node changes its hardware address, for example, and it is on the same subnet as the local station, then the table entry would have to be manually cleared for that IP node. Again, if this causes problems, we may have to implement a time out for the IPARP table entries. Both this timeout and the queuing of datagrams awaiting ARP responses should probably be implemented at the same time.

It is hoped that 254 table entries, which means 254 IP addresses, will be enough for the local stations to keep track of. If it is not, then this would be another reason to implement a timeout on the IPARP table entries.

With no timeout of IPARP table entries, it is still ok for a station to change its hardware address. if that station sends a request to the local station. The hardware address used in the request message will update the IPARP table entry for that same IP address. If the change is in the IP address, a new entry will be automatically added to the table. If an IPARP table entry is cleared manually, the entry will be available for re-use.

Installation

When installing IP support for the first time, install the table #28 as 256 entries of 16 bytes each in the system table directory. After reset, the table will be initialized, but no IP address will be in its header. Install the IP address, subnet mask, and the default gateway address. Reset again. One other important item: install the SAP table entry#2 as $AA, or no SNAP (IP) frames can be received!